Grammar and Reference

The tools and resources for writing and editing on Linux-based systems include spell checkers, dictionaries, and reference files. This chapter shows methods for using them.

11.1 Spell Checking

There are several ways to spell-check text and files on Linux; the following recipes show how to find the correct spellings of particular words and how to perform batch, interactive, and Emacs-based spell checks.

The *system dictionary* file, `/usr/dict/words`,[1] is nothing more than a word list (albeit a very large one), sorted in alphabetical order and containing one word per line. Words that are correct regardless of case[2] appear in all lowercase letters, and words that rely on some form of capitalization in order to be correct (such as proper nouns) appear in that form. All of the Linux spelling tools use this text file to check spelling; if a word does not appear in the dictionary file, it is considered to be misspelled.

⇒ To find out how many words come in the user dictionary on your system, type:

```
$ wc -l /usr/dict/words ⟨RET⟩
```

NOTES: If you are using the wrong word to begin with, none of the computerized spell-check tools will correct this error—for example, if you have "`there`" when you mean "`their`," the computer cannot catch it (yet!).

11.1.1 Finding the Correct Spelling of a Word

If you're unsure whether or not you're using the correct spelling of a word, use `spell` to find out. `spell` reads from the standard input and outputs any words not found in the system dictionary—so if a word is misspelled, it will be echoed back on the screen after you type it.

---

[1] On an increasing number of systems, this file is being replaced with the newer `/usr/share/dict/words`; administrators should make a symbolic link from this to the shorter, preferred form.

[2] In other words, they are correct whether they appear entirely in lowercase letters, capitalized, or entirely in uppercase letters.

⇒ For example, to check whether the word "occurance" is misspelled, type:

```
$ spell ⟨RET⟩
occurance ⟨RET⟩
occurance
⟨CTRL⟩-⟨D⟩
$
```

In this example, spell echoed the word "occurance," meaning that this word was not in the system dictionary and therefore was quite likely a misspelling. Then, ⟨CTRL⟩-⟨D⟩ was typed to exit spell.

11.1.2 Listing the Misspellings in Text

To output a list of misspelled words in a file, give the name of the file to check as an argument to spell. Any misspelled words in the file are output, each on a line of its own and in the order that they appear in the file.

⇒ To spell-check the file fall-lecture.draft, type:

```
$ spell fall-lecture.draft ⟨RET⟩
occurance
willl
occurance
$
```

In this example, three words are output: "occurance," "willl," and "occurance" again, meaning that these three words were found in fall-lecture.draft, in that order, and were not in the system dictionary (and so were probably misspelled). Note that the misspelling "occurance" appears twice in the file.

To correct the misspellings, you could then open the file in your preferred text editor and edit it. Later in this section, I'll describe an interactive spell checker that allows you to correct misspellings as they are found. Still another option is to use a text editor with spell-checking facilities built in, such as Emacs.

⇒ To spell-check the file fall-lecture.draft, and output any possibly misspelled words to a file fall-lecture.spelling, type:

```
$ spell fall-lecture.draft > fall-lecture.spelling ⟨RET⟩
```

In this example, the standard output redirection operator (>) is used to redirect the output to a file (see Recipe 3.2.2 [Redirecting Output to a File], page 68).

To output an *alphabetical* list of the misspelled words, pipe the output to `sort`; then pipe the sorted output to the `uniq` filter to remove duplicates from the list (`uniq` removes duplicate adjacent lines from its input, outputting the "unique" lines).

⇒ To output a sorted list of the misspelled words that are in the file `fall-lecture.draft`, type:

```
$ spell fall-lecture.draft | sort | uniq ⟨RET⟩
```

11.1.3 Keeping a Spelling Word List

The stock American English dictionary installed with Linux-based systems includes over 45,000 words. However large that number may seem, a lot of words are invariably left out—including slang, jargon, and some proper names.

You can view the system dictionary as you would any other text file, but users never edit this file to add words to it.[3] Instead, you add new words to your own *personal dictionary*, a file in the same format as the system dictionary, but kept in your home directory as the file `~/.ispell_default`.[4]

A user can have his own personal dictionary; the spelling commands discussed in this chapter automatically use your personal dictionary, if you have one, in addition to the system dictionary.

You build your personal dictionary using the `i` and `u` options of `ispell`, which insert words into your personal dictionary. Use these options either with the stand-alone tool or with the various `ispell` Emacs functions (see Recipe 11.1.4 [Interactive Spell Checking], page 278, and Recipe 11.1.5 [Spell Checking in Emacs], page 280).

⇒ To find out how many words you have in your personal dictionary, type:

```
$ wc -l .ispell_default ⟨RET⟩
```

NOTES: You can also add (or remove) words by manually editing the file with a text editor, but take care to keep the list in alphabetical order!

---

[3] If a word is reasonably universal, you may, of course, contact the global maintainers of `wenglish` or other appropriate packages, and try to convince them that said word ought to be included.

[4] On newer systems, this file is sometimes replaced by `~/aspell_default`.

Over time, personal dictionaries begin to look very personal, as a reflection of their owners; Gregory Cosmo Haun made a work of conceptual art by photographing the portraits of a dozen users superimposed with listings of their personal dictionaries [`http://www.reed.edu/~cosmo/art/DictPort.html`].

11.1.4 Interactive Spell Checking

GNU Aspell
DEB: `aspell`
RPM: `aspell`
WWW: `http://aspell.net/`

or

Ispell
DEB: `ispell`
RPM: `ispell`
WWW: `http://fmg-www.cs.ucla.edu/geoff/ispell.html`

Use `ispell` to spell check a file *interactively*, so that every time a misspelling is found, you're given a chance to replace it then and there.[5]

⇒ To interactively spell-check `fall-lecture.notes`, type:

```
$ ispell fall-lecture.notes ⟨RET⟩
```

When you type this, `ispell` begins checking the file. It stops at the first misspelling it finds, as in Figure 11-1.

On the top line of the screen, `ispell` displays the misspelled word, followed by the name of the file. Underneath this is the sentence in which the misspelling appears, with the word in question highlighted. Following this is a list of suggested words, each offset by a number—in this example, `ispell` has only one suggestion: "`lectures`."

To replace a misspelling with a suggested word, type the number that corresponds to the suggested word (in this example, you would type *0* to replace the misspelling with "`lectures`"). You only need to type the number of your selection—a ⟨RET⟩ is not required.

You can also type a correction yourself; this is useful when `ispell` either offers no suggestions, or when it does and the word you want is not one of them. To do this, type *r* (for "replace") and then type the replacement word, followed by ⟨RET⟩.

[5] `aspell` is designed to be a drop-in replacement for `ispell`, with greatly improved suggestion algorithms. If your system has `aspell`, use it instead; it otherwise works like `ispell`.

Sometimes, `ispell` will question a word that you may not want to count as a misspelling, such as proper names and the like—words that don't appear in the system dictionary. There are a few things you can do in such cases, as follows.

To accept a misspelled word as correct for the current `ispell` session only, type `a`; from then on during the current session, this word will be considered correct.

```
lecutres              File: fall-lecture.notes

The focus of my series of lecutres this fall are the aspects of the novel.
 0: lectures



[SP] <number> R)epl A)ccept I)nsert L)ookup U)ncap Q)uit e(X)it or ? for help
```

Figure 11-1. A misspelling caught by `ispell`.

If, however, you want `ispell` (and `spell`, and all other tools that access the system dictionary) to remember this word as being correct for this and all future sessions, insert the word in your own personal dictionary. Type `u` to insert a copy of the word *uncapitalized*, in all lowercase letters—this way, even if the word is capitalized at the beginning of a sentence, the lowercase version of the word is saved. From then on, in the current `ispell` session and in future sessions, this word will be considered correct whether it appears entirely in lowercase letters, capitalized, or entirely in uppercase letters.

When case is important to the spelling—for example, in a word that is a proper name such as "`Seattle`," or a word with mixed case, such as "`DeSalle`"—type `i` to insert a copy of the word in your personal dictionary with its case *just as it appears*; this way, words spelled with the same letters but with different cases will be considered misspellings.

When `ispell` finishes spell-checking a file, it saves its changes to the file and then exits. It also makes a copy of the original file, without the changes applied; this file has the same name as the original but with `.bak` added to the end—in our example, the backup file is called `fall-lecture.notes.bak`.

This is useful if you regret the changes you've made and want to restore the file to how it was before you mucked it up—just remove the spell-checked file and then rename the `.bak` file to its original name.

The following table is a reference to the `ispell` key commands, listing the keys and describing their actions.

⟨SPACEBAR⟩	Accept misspelled word as correct, but only for this particular instance.
number	Replace misspelled word with the suggestion that corresponds to the given number.
`?`	Display a help screen.
`a`	Accept misspelled word as correct for the remainder of this `ispell` session.
`i`	Accept misspelled word as correct and add it to your private dictionary with the capitalization as it appears.
`l`	Look up words in the system dictionary according to a pattern you give.
`q`	Quit checking and restore the file to how it was before this session.
`r`	Replace misspelled word with a word you type.
`u`	Accept misspelled word as correct and add it to your private dictionary in all lowercase letters.
`x`	Save changes made so far, and then stop checking this file.

11.1.5 Spell Checking in Emacs

Emacs has several useful commands for spell-checking. The `ispell-word`, `ispell-region`, and `ispell-buffer` functions, as you might guess from their names, use the `ispell` command inside Emacs to check portions of the current buffer.[6]

The first command, `ispell-word`, checks the spelling of the word at point; if there is no word at point, it checks the first word to the left of point. This

[6] On many newer systems, `aspell` is used in `ispell`'s place.

command has a keyboard shortcut, ⟨ALT⟩-⟨$⟩. The second command, `ispell-region`, checks the spelling of all words in the currently selected region of text. The third command, `ispell-buffer`, checks the spelling of the entire buffer.

⇒ Here are some ways to use this.

- To check the spelling of the word at point, type:
  ⟨ALT⟩-⟨X⟩ *ispell-word* ⟨RET⟩
- To check the spelling of all words in the currently selected region of text, type:
  ⟨ALT⟩-⟨X⟩ *ispell-region* ⟨RET⟩
- To check the spelling of all words in the current buffer, type:
  ⟨ALT⟩-⟨X⟩ *ispell-buffer* ⟨RET⟩

Another useful Emacs spelling feature is `flyspell-mode`. When this mode is set in a buffer, any misspelled words in the buffer are highlighted. This mode is useful when you are writing a first draft, because it lets you catch misspellings as you type them.

⇒ To turn on `flyspell-mode` in a buffer, type:
⟨ALT⟩-⟨X⟩ *flyspell-mode* ⟨RET⟩

NOTES: This mode is a toggle; run it again to turn it off.

To correct a word in `flyspell-mode`, click and release the middle mouse button on the word to pull up a menu of suggestions; you then use the mouse to select the replacement word or add it to your personal dictionary.

If there are words you frequently misspell, you can define abbrevs for them (see Recipe 10.1.5 [Making Abbreviations in Emacs], page 242). Then, when you type the misspelled word, Emacs will automatically replace it with the correct spelling.

Finally, if you prefer the sparse, non-interactive interface of `spell`, you can use the Emacs interfaces to that command instead: `spell-word`, `spell-region`, and `spell-buffer`. When any of these functions find a misspelling, they prompt for a replacement in the minibuffer but do not offer suggestions or provide any of `ispell`'s other features.

11.2 Using Dictionaries

WordNet
DEB: `wordnet`
`wordnet-base`
WWW: `http://www.cogsci.princeton.edu/~wn/`

The term *dictionary* on Linux systems generally refers to one of two things: the traditional UNIX-style dictionary, which is an alphabetically sorted word list containing no actual definitions, and the newer database-style dictionary that contains the headwords as well as their definitions. The latter is the kind of thing most people mean when they talk about dictionaries. (When most UNIX folk talk about dictionaries, however, they almost always mean the former.)

WordNet is a lexical reference system in the form of a database containing thousands of words arranged in synonym sets. You can search the database and output the results in text with the `wn` tool or the `wnb` X client (the "WordNet browser").

Use of the X client is fairly straightforward—type a word in the dialog box near the top of the screen, followed by ⟨RET⟩, to get its definitions, which are displayed in the large output window underneath the dialog box.

For example, when you do a search for the definition of the word "`browse`," the WordNet browser will look like Figure 11-2.

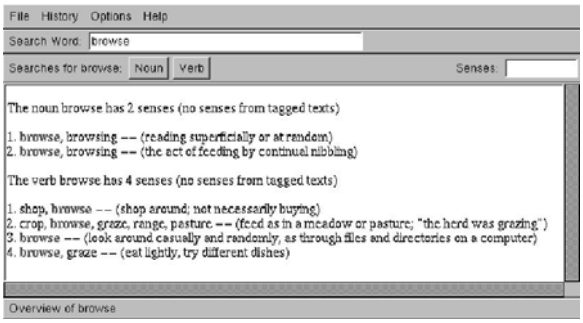

Figure 11-2. The WordNet browser.

Between the dialog box and the output window, there are menus for searching for synonyms and other word senses. A separate menu is given for each part of speech a word may have; in the preceding example, the word "`browse`" can be either a noun or a verb, so two menus are shown.

To get a list of all word sense information available for a given word, run `wn` with the word as an argument. This outputs a list of all word sense information available for the word, with each possible sense preceded with the name of the option to use to output it.

⇒ To output a list of word senses available for the word "`browse`," type:

```
$ wn browse ⟨RET⟩
```

The following sections show how to use `wn` on the command line.

NOTES: For more information on WordNet, consult the `wnintro man` page (see Recipe 2.8.4 [Reading a Page from the System Manual], page 46).

11.2.1 Listing Words That Match a Pattern

There are several ways to search for and output words from the system dictionary.

Use `look` to output a list of words in the system dictionary that begin with a given string—this is useful for finding words that begin with a particular phrase or prefix. Give the string as an argument; it is not case-sensitive.

⇒ To output a list of words from the dictionary that begin with the string "`homew`," type:

```
$ look homew ⟨RET⟩
```

This command outputs words like "`homeward`" and "`homework`."

Since the system dictionary is an ordinary text file, you can also use `grep` to search it for words that match a given pattern or regular expression (see Recipe 14.3 [Matching Patterns of Text], page 335).

⇒ Here are some ways to use this.

- To list all words in the dictionary that contain the string "`dont`," regardless of case, type:

```
$ grep -i dont /usr/dict/words ⟨RET⟩
```

- To list all words in the dictionary that end with "`ing`," type:

```
$ grep ing^ /usr/dict/words ⟨RET⟩
```

- To list all of the words that are composed only of vowels, type:

```
$ grep -i '^[aeiou]*$' /usr/dict/words ⟨RET⟩
```

To find some words that rhyme with a given word, use `grep` to search `/usr/dict/words` for words ending in the same last few characters as the word they should rhyme with (see Recipe 14.4.2 [Matching Lines Ending with Certain Text], page 343).

⇒ To output a list of words that rhyme with "friend," search /usr/dict/words for lines ending with "end":

```
$ grep 'end$' /usr/dict/words ⟨RET⟩
```

Finally, to do a search on the WordNet dictionary, use wn with one of the -grep options. When you give some text to search for as an argument, this command does the equivalent search as look, but only the particular kind of word sense you specify is searched: -grepn searches nouns, -grepv searches verbs, -grepa searches adjectives, and -grepr searches adverbs. You can combine options to search multiple word senses.

⇒ Here are two ways to use this.

- To search the WordNet dictionary for nouns that begin with "homew," type:

```
$ wn homew -grepn ⟨RET⟩
```

- To search the WordNet dictionary for both nouns and adjectives that begin with "homew," type:

```
$ wn homew -grepn -grepa ⟨RET⟩
```

11.2.2 Listing the Definitions of a Word

To list the definitions of a word, give the word as an argument to wn, followed by the -over option.

⇒ To list the definitions of the word "slope," type:

```
$ wn slope -over ⟨RET⟩
```

NOTES: If you look up dictionary definitions frequently enough, it is handy to have an alias of "def" defined as wn $1 -over (see Recipe 3.6.1 [Calling a Command by Some Other Name], page 83).

11.2.3 Listing the Synonyms of a Word

A *synonym* of a word is a different word with a similar meaning that can be used in place of the first word in some context. To output synonyms for a word with wn, give the word as an argument, followed by one of the following options: -synsn for nouns, -synsv for verbs, -synsa for adjectives, or -sysnr for adverbs.

⇒ Here are two ways to use this.

- To output all of the synonyms for the noun "break," type:

```
$ wn break -synsn ⟨RET⟩
```

- To output all of the synonyms for the verb "break," type:

  $ *wn break -synsv* ⟨RET⟩

11.2.4 Listing the Antonyms of a Word

An *antonym* of a word is a different word that has the opposite meaning of the first in some context. To output antonyms for a word with wn, give the word as an argument, followed by one the following options: -antsv for verbs, -antsa for adjectives, or -antsr for adverbs.

⇒ To output all of the antonyms for the adjective "sad," type:

  $ *wn sad -antsa* ⟨RET⟩

11.2.5 Listing the Hypernyms of a Word

A *hypernym* of a word is a related term whose meaning is more general than the given word. (For example, the words "mammal" and "animal" are hypernyms of the word "cat.")

To output hypernyms for a word with wn, use one of the following options: -hypen for nouns or -hypev for verbs.

⇒ To output all of the hypernyms for the noun "cat," type:

  $ *wn cat -hypen* ⟨RET⟩

11.2.6 Checking Online Dictionaries

Dict
DEB: dict
RPM: dict
WWW: http://www.dict.org/

The DICT Development Group has a number of free dictionaries on its Web site [http://www.dict.org/]. On that page, you can look up words (including using a thesaurus and other searches) from a dictionary that contains over 300,000 headwords, or you can make a copy of its dictionary for use on your own system. A dict client exists for accessing DICT servers and outputting definitions locally; this tool is available in the dict package.

There are a number of specialized dictionaries available from the DICT Development Group as well. These dictionaries are plain text files. One such dictionary is called FILE, *The Free Internet Lexicon and Encyclopedia*. It is an effort to build a free, open source collection of modern-word, idiom, and

jargon dictionaries. FILE is a volunteer effort and depends on the support of scholars and lexicographers; the DICT pages contain information on how to help contribute to this worthy project.

11.3 Checking Grammar

Diction
DEB: `diction`
WWW: `http://www.gnu.org/software/diction/diction.html`

Two venerable UNIX tools for checking writing have recently been made available for Linux-based systems: `style` and `diction`.

Old-timers probably remember these names—the originals came with AT&T UNIX as part of the much-loved "Writer's Workbench" (WWB) suite of tools back in the late 1970s and early 1980s.[7]

AT&T "unbundled" the Writer's Workbench from its UNIX Version 7 product, and as the many flavors of UNIX blossomed over the years, these tools were lost by the wayside—eventually becoming the stuff of UNIX lore.

In 1997, Michael Haardt wrote new Linux versions of these tools from scratch. They support both the English and German languages, and they're now part of the GNU Project.

Two additional commands that were part of the Writer's Workbench have long been standard on Linux: `look` and `spell`, described previously in this chapter.

The following are recipes that use either `diction` or `style`.

11.3.1 Checking Text for Misused Phrases

Use `diction` to check for wordy, trite, clichéd, or misused phrases in a text. It checks for all the kind of expressions William Strunk warned us about in his *Elements of Style* [`http://www.bartleby.com/141/`].

According to *The UNIX Environment* (see Appendix D [References for Further Interest], page 731), the `diction` tool that came with the old Writer's Workbench just *found* the phrases, and a separate command called `suggest` would output suggestions. In the GNU version that works for Linux systems, both functions have been combined in the single `diction` command.

[7] There was also a set of tools for formatting text called the "Documenter's Workbench" (DWB), and there was a planned "Reader's Workbench"; today, we can only guess at what that might have been.

In GNU `diction`, the words or phrases are enclosed in brackets "`[like this]`." If `diction` has any suggested replacements, it gives them preceded by a right arrow, "`-> like this`."

When checking more than just a screenful of text, you'll want to pipe the output to `less` so that you can peruse it on the screen (see Recipe 9.1 [Perusing Text], page 211), or pipe the output to a file for later examination.

⇒ Here are two ways to use this.

- To check the file `dissertation` for clichés or other misused phrases, type:

  ```
  $ diction dissertation | less ⟨RET⟩
  ```

- To check the file `dissertation` for clichés or other misused phrases, and write the output to a file called `dissertation.diction`, type:

  ```
  $ diction dissertation > dissertation.diction ⟨RET⟩
  ```

If you don't specify a file name, `diction` reads text from the standard input until you type ⟨CTRL⟩-⟨D⟩ on a line by itself. This is especially useful when you want to check a single sentence, as in Figure 11-3.

```
$ diction ⟨RET⟩
Let us ask the question we wish to state. ⟨RET⟩
(stdin):1: Let us [ask the question -> ask]
[we wish to state -> (cliche, avoid)].
⟨CTRL⟩-⟨D⟩
$
```

Figure 11-3. Checking a sentence with `diction`.

To check the text of a Web page, use `lynx` with the `-dump` and `-nolist` options to output the plain text of a given URL, and pipe this output to `diction`. (If you expect there to be a lot of output, add another pipe at the end to `less` so you can peruse it.)

To peruse the URL `http://www.westegg.com/cliche/random.cgi` with markings for possible wordy and misused phrases, type (all on one line):

```
$ lynx -dump -nolist http://www.westegg.com/cliche/random.cgi |
diction | less ⟨RET⟩
```

NOTES: To check text for overused words, use the method described in Recipe 12.2.4 [Counting Word Occurrences in Text], page 299.

11.3.2 Checking Text for Doubled Words

One of the things that `diction` looks for is doubled words—words repeated twice in a row. If it finds such a sequence, it encloses the second member of the doubled pair in brackets, followed by a right arrow and the text "`Double word`," like "`this [<i>this -> Double word.]`."

To check a text file for doubled words *only*, and not for any of the other things `diction` checks, use `grep` to find only those lines in `diction`'s output that contain the text "`Double word`," if any.

⇒ To output all lines containing double words in the file `dissertation`, type:

```
$ diction dissertation | grep 'Double word' ⟨RET⟩
```

11.3.3 Checking Text for Readability

The `style` command analyzes the writing style of a given text. It performs a number of readability tests on the text and outputs their results, and it gives some statistical information about the sentences of the text. Give as an argument the name of the text file to check.

⇒ To check the readability of the file `dissertation`, type:

```
$ style dissertation ⟨RET⟩
```

Like `diction`, `style` reads text from the standard input if no text is given—this is useful for the end of a pipeline, or for checking the writing style of a particular sentence or other text you type.

The sentence characteristics of the text that `style` outputs are as follows:

- Number of characters
- Number of words, their average length, and their average number of syllables
- Number of sentences and average length in words
- Number of short and long sentences
- Number of paragraphs and average length in sentences
- Number of questions and imperatives

The various readability formulas that `style` uses and outputs are as follows:

- Kincaid formula, originally developed for Navy training manuals; a good readability test for technical documentation

- Automated Readability Index (ARI)
- Coleman-Liau formula
- Flesch Reading Ease Score, which gives an approximation of readability from 0 (difficult) to 100 (easy)
- Fog Index, which gives a school-grade reading level
- WSTF Index, a readability indicator for German documents
- Wheeler-Smith Index, Lix formula, and SMOG-Grading tests, all readability indicators that give a school-grade reading level

11.3.4 Checking Text for Difficult Sentences

To output just the "difficult" sentences of a text, use `style` with the `-r` option followed by a number; `style` will output only those sentences whose Automated Readability Index (ARI) is greater than the number you give.[8]

⇒ To output all sentences in the file `dissertation` whose ARI is greater than a value of 20, type:

```
$ style -r 20 dissertation ⟨RET⟩
```

11.3.5 Checking Text for Long Sentences

Use `style` to output sentences longer than a certain length by giving the number of words as an argument to the `-l` option.

⇒ To output all sentences longer than 14 words in the file `dissertation`, type:

```
$ style -l 14 dissertation ⟨RET⟩
```

11.4 Using Reference Files

There are reference works and other informative text files that you can install on your system; the following recipes describe some of the more interesting and useful ones that are readily available.

[8] To get an idea how the ARI ranks text, see its rankings for various popular Web sites at `http://www.readability.info/commonscores.shtml`.

11.4.1 Consulting Word Lists and Helpful Files

Miscfiles
DEB: `miscfiles`
WWW: `ftp://ftp.gnu.org/pub/gnu/miscfiles/miscfiles-1.1.tar.gz`

The GNU Miscfiles collection is a group of text files containing various facts and reference material, such as common abbreviations, telephone area codes, and English connective phrases.

The files are stored in the `/usr/share/misc` directory, and they are all compressed; use `zless` to peruse them (see Recipe 9.1 [Perusing Text], page 211).

The following table lists the files as they appear in `/usr/share/misc` and describes their contents.

`GNU-manifesto.gz`	The GNU Manifesto.
`abbrevs.talk.gz` `abbrevs.gen.gz`	Collections of common abbreviations used in electronic communication. (This is the place to look to find the secrets of TTYL and LOL.)
`airport.gz`	List of three-letter city codes for some of the major airports. The city code is useful for querying the National Weather Service computers to get the latest weather report for your region.
`ascii.gz`	A chart of the ASCII character set.
`birthtoken.gz`	The traditional stone and flower tokens for each month.
`cities.dat.gz`	The population, political coordinates (nation, region), and geographic coordinates (latitude, longitude) of many major cities.
`inter.phone.gz`	International country and city telephone codes.
`languages.gz`	Two-letter codes for languages, from ISO 639.
`latin1.gz`	A chart of the extended ASCII character set, also known as the ISO 8859 ("Latin-1") character set.
`mailinglists.gz`	Description of all the public GNU Project mailing lists.

`na.phone.gz`	North American (+1) telephone area codes.
`operator.gz`	Precedence table for operators in the C programming language.
`postal.codes.gz`	Postal codes for U.S. and Mexican states and Canadian provinces.
`us-constitution.gz`	*The Constitution of the United States of America* and its twenty-seven Amendments (the first ten are the *Bill of Rights*). On Debian systems, this file is placed in a directory named `/usr/share/state`.
`us-declaration.gz`	*The Declaration of Independence of the Thirteen Colonies*. On Debian systems, this file is placed in a directory named `/usr/share/state`.
`rfc-index.txt`	Indexes of Internet standardization Request For Comments (RFC) documents. On Debian systems, this file is placed in `/usr/share/rfc`.
`zipcodes.gz`	U.S. five-digit Zip Codes.

But `miscfiles` is not the only reference package available for Linux; other related packages include the following:

doc-iana	Internet protocol parameter registry documents, as published by the Internet Assigned Numbers Authority. DEB: `doc-iana`
Jargon File	The "Jargon File" is the definitive dictionary of hacker slang, and goes back decades. Might be considered somewhat dated today; is no longer distributed as a single file. DEB: `jargon-text` RPM: `jargon` WWW: `http://www.jargon.org/`
V.E.R.A.	Extensive list of computer acronyms. DEB: `vera` RPM: `vera` WWW: `ftp://ftp.gnu.org/gnu/vera/`

NOTES: The official GNU `miscfiles` distribution also includes the Jargon File and the `/usr/dict/words` dictionary file, which are available in separate packages for Debian, and are removed from the Debian `miscfiles` distribution. On Debian systems, `/usr/dict/words` is part of the standard spelling packages, and the Jargon File comes in the optional `jargon` package and installs in `/usr/share/jargon`.

11.4.2 Translating Common Acronyms

Bsdgames
DEB: `bsdgames`
RPM: `bsd-games`
WWW: `ftp://metalab.unc.edu/pub/Linux/games/`

Use `wtf` to see what an acronym stands for. This is useful for decoding the kind of acronyms commonly used in Usenet chatter and other online chat forums. If it doesn't know the answer, it checks with `whatis` before giving up (see Recipe 2.8.2 [Getting a Description of a Program], page 46), so it will also tell you what any tool or program installed on the system is.

Give the acronym to translate as an argument to `wtf`, optionally preceded by "`is`."

⇒ To translate the acronym "LOL," type:

```
$ wtf lol ⟨RET⟩
```

Typing *`wtf is lol`* produces an identical result.

NOTES: While `wtf` is useful for decoding the acronyms of online writing, it doesn't know anything outside of this scope. So it can't tell you, for instance, what LASER stands for (but the WordNet dictionary can—see Recipe 11.2 [Using Dictionaries], page 282), although it will tell you all about how YMMV even when you do RTFB, so you see it is NBD.

Other collections of acronyms are available with the `miscfiles` collection (see Recipe 11.4.1 [Consulting Word Lists and Helpful Files], page 290).